Once more into the breech: computer literacy and the humanities

Roddy, Kevin P.

ONCE MORE INTO THE BREECH: COMPUTER LITERACY AND THE HUMANITIES

Kevin P. Roddy (*)

On occasion, computer-oriented humanists and social scientists are asked to serve on committees setting institutional computer policy. Though many such committees are often neither necessary nor productive, those which are now studying the encouragement of "computer literacy" an exception: I would like to argue that they are vitally important to the future of social history, and to all culture studies. The decisions made now will have a direct bearing on the preparedness of the next generation of scholars, and it is very much in our interest to define those qualities which characterize computer literacy in our field. The following paper represents a contribution to this effort.

In the past, our attitude toward the computer literacy movement has largely been one of suspicion. In general, though we have attempted to enlighten individuals students about computers, we have been reluctant to impose our attitudes on the institution. The phrase "computer literacy" itself is imprecise and emotionally tinged, and those promoting it – elected officials and institutional administrators - have not usually been known for their experience with computers. Some three years ago, when the phrase became popular, my colleagues and I immediately detected the logical flaw in the "literacy" metaphor; and we feared that, in the United States at least, such literacy would necessarily be provided at the expense of traditional studies, notably the acquisition of foreign languages. These concerns are still real and justified: but, as I suggested above, we also have a responsibility to participate in nurturing literacy, even though we might not have been present at its birth.

I feel that our participation is all of more necessary because computer literacy is almost always discussed on the basis of two unexamined assumptions: that the computers are microcomputers, and that literacy means programming. In many American Universities, with surprising uniformity, the desire to foster computer literacy has translated into ambitious programs to place a microcomputer on the desk of every member of the faculty, and to arrange that every student have access to one. But from my perspective, however much micros are used, and however handy they may be, I find it difficult to consider such use "literacy." The issue is not one of power and sophistication; these are ephemeral technical issues which are being resolved elsewhere. My specific objection to micros is that they represent a return to isolated scholarship, just when computers were making it easy and profitable for us to communicate, to share information, methodology, and even programs. Furthermore, to speak on a more pragmatic level, one has only to say the words "Sinclair," "Osbourne," "DEC Rainbow," "IBM Peanut," and even "CPM" to realize how volatile the micro industry has been. It is an irony which might after all teach the best lesson of all about computers, if the micro that a first-year student had been forced to buy became obsolete before the third year.
The second general assumption has been that computer literacy should be defined as programming, in and of itself. The point is complex, and requires considerable discussion, but at present it might be helpful to

---

(') Address all communications to : Kevin P. Roddy, Department of Medieval Studies, University of California at Davis, Davis, California, 95616, USA.

consider what programming exercises are meant to accomplish, and whether or not programming, in and of itself, is both more specific and more advanced than might reasonably be encompassed under the heading "literacy." The simple explanation is that programming is meant to program; that is, students must learn to control the computer. We are, however, then faced with two contrasting situations: the tendency of programmers to cling to the language they were taught, and the ease with which a programming language, if not used, is forgotten. In the United States, for example, those students entering professions which require programming - engineering and applied science - are often taught FORTRAN in their first years; for the next two years, however, they do not program in their courses, since these are introductory in nature. In their third year, when they begin to research and thus need programming, the experience has been that they have forgotten everything. My point is that, if programming in itself constituted literacy, one would not expect it to reside in short-term memory. If, moreover, knowing a programming language were really as critical a skill as many clain, then American universities should be teaching COBOL, which is the language of choice for our bureaucracies and financial institutions.

What, then, is computer literacy, if it does not need micros and will not, on at least an introductory level, involve programming? For my part, computer literacy is a clear understanding of the computer's potentials. I realize that this makes me, along with a number of my colleagues, illiterate. But if we examine that remarkable word "literacy," we find that it has so positive a connotation because some of the greatest minds of our civilization, Martin Luther and Thomas Jefferson among them, fostered literacy, not merely as a means of mastering letters as much as an access to the rich potential of written language. Teaching such literacy has been accomplished by clearly identifying the rationale of access, the basic principles, simultaneously with a discussion of their application. The literacy metaphor may then have merit, if computer literacy follows the same route, to an understanding of what the computer can now make accessible. To put it another way, the would-be computer literate first needs answers to three basic questions: what can a computer do (and, by extension, what can't it do)? what will it cost to do it? what is absolute and what is flexible about computing - what, in other words, can be changed and what cannot?

1 would like to suggest that any curriculum designed to foster computer literacy, especially for humanities students, first respond to these three questions. It may be that the questions themselves are trivial, and the answers obvious; I think not. It may well be that the questions are preposterous, and the answers impossible; again I think not. If the former is true, perhaps we should belabor the obvious in the first minute of the first class, before moving to important issues; if the latter is true, perhaps this in itself is worth nothing. If the three questions are neither trivial nor preposterous, then they might fruitfully be discussed; in the next few pages, I should like to take that opportunity.

In my experience, the question what can a computer do raises some important issues, but only if the students are given sufficient time to consider it in all of its implications. Computers, we say, are capable of storing vast quantities of information. How, then, the students ask, does the user keep track of all? What is the chance that an obsolete version will supplant a current version? In response we are forced to admit that this has been a problem: far more energy hass been expended in entering

data than in making it available. Can one, students may persist in asking, put a book on the computer? Certainly, we answer, no longer quite sure of ourselves, though this is not exactly a simple matter. Can one then do close and rigorous analysis on such a text? Yes, though this may depend on factors beyond the capacity of the computer. An example might be the Domesday Book Project at the University of California, Santa Barbara, which has largely completed encoding that great and uniquely detailed monument to medieval record-keeping. Unfortunately, the Project proceeded for many years unaware of a similar program at the University of Hull to enter and edit the entire book. The two projects are now cooperating, but one can imagine how useless the Santa Barbara material would have been if it remained based on an obsolete text. This is a kind of post-facto literacy. The fundamental importance of the question, what can a computer do, has been emphasized over and over by such experienced realists as David Vance of the Museum Computer Network, and Michael Preston of the University of Colorado, a foremost authority on literary concordances. A recent editorial in the bulletin Scholarly Communication pointed out, to my complete surprise, what should have been perfect obvious: computers are better at telling differences than at identifying similarities. This, for example, makes the computer excellent for proof-reading and terrible at spelling. We need more such basic insight, and this means spending more time acquiring it.

Within this section of my computer literacy course I would like to add a section on pathology. If one goes to the many relevant conferences one will hear little of failure, though it must exist. Usually, disasters are mentioned only as introductions to successes. In such cases, the cause seems to have been insufficient technological expertise; now that the project director has taught himself, there will be no more failure. I for my part disagree, and suspect that most failures are not technical, but rather arise from unrealistic (that is, illiterate) notions about what the computer can do. A computer will not improve a badly planned project, a computer will not guess the right answer, and a computer will not raise the dead. Perhaps from all of this the lesson will emerge that an hour's conference, involving every single participant, should be scheduled for each day, whether it's needed or not; if so, I for one would be very glad to learn it.

The second question, how much does it cost, cannot be considered trivial, but some might consider it irrelevant. "Cost" does not simply entail money, which in computing has an otherwordly aspect; rather there are the other, more elusive expenses: human resources, machine resources, and that most elusive expense of all, time. But it might still be argued that it is irrelevant to consider even these when using computers: the nature of pure research and development is such that it will never be "cost effective". This would unfortunately ignore the realities of computer projects, which do founder because a key member of the staff has left, or because not enough vision was used in the choice of hardware, peripherals, and language, or because there was simply no time to accomplish the objectives, or, and this happens, because the funding agency lost patience. That research in computing, whether in the humanities or elsewhere, should proceed without these limitations goes without saying; but, since we do labor under them, it does seem more literate to take them into account. And I do not feel that this is altogether evil. If, for example, a project will take longer than anyone imagines, would it not be wiser to plan for several intermediate stages, each with an identifiable and generally useful product? Along the way, for instance, to a text

analysis program there might be a portable routine for recognizing words. Such a routine must necessarily be both more general and less efficient than that devoted to the specific purposes at hand. But it will also be something to salvage it fortune proves unkind. To cite a common situation, for certain projects I have found it very easy to enlist the help of computer freaks, "hackers". And I have also found it easy to lose that help as the novelty wears off, and the project has to compete with order, newer demands on their expertise. It would be extremely valuable to future scholars in the field to described such syndromes, as remote from systems analysis and programming as they may be, explain them, and recommend appropriate solutions.

The third question, how flexible can the computer be, is the most difficult, and it may be here that the argument for programming carries the most weight. But I note to my sadness that many of the humanities in the United States who have moved deeply into programming have done so at a cost to their research. Manipulation of material through the computer is undeniably more exciting than reading three hundred articles on folk marriage, especially if most of the articles derive from each other. But how does one then maintain perspective? One of the really interesting results of many of the concordances produced recently is the discovery that most editions are not very good after all. One would think that attention would certainly be shifted to computer-aided editions. But perhaps because editions demand arduous work and comparatively little programming, this shift has not occurred. Concordances continue to be produced, and humanists continue to write concordance programs. While I admit that concordances are needed, and that one person's program will not work for another, I would still suggest that the solution does not lie in learning programming, but rather in finding or in encouraging those programs which are sufficiently protean to accomplish the task, and various similar general tasks as well.

I have already described one aspect which might characterize such a program: a portable modularity. A second characteristic might be termed "malleability," the power to intrude on the raw data or redefine its format, without endangering the totality of the process. Specifically, I would like to suggest that the project director, in designing the system, identify those critical junctures at which the individual user is likely to intervene, and at such junctures allow modification in an completely unrestricted way. This is a different from "menu-driven", which after all expects the impossible, that the programmer anticipate every conceivable need. Rather, in my judgment, the user should be offered the ability to manipulate certain functions, in respect to the contents and type of information, as well as in respect to its format. I freely admit that the ability to effect these changes borders on the ability to program. But however much it does presume a certain level of sophistication from the user, the sophistication is basically editorial in nature, and therefore one which should be within the abilities of any humanist. More importantly, the feature of malleability concerns the humanist's area of expertise, as humanist. That is, we are supposed to know what we want to put in, what we want to take out, and how we want to look at it. To cite an example in my experience, most bibliographic data bases are arranged logically, with the author in one field, the book title in another, and so on. Humanities software like Nota bene offer the option of reformatting such data according to the stylistic rules of the Modern Languages Association, or the American Psychological Association, and so on. It has not been until version 2, however, which was due to be released in May,

that the literature promises the ability to modify these routines; this is sixteen months after version 1 originally appeared. In view of the fact that no one has precisely the same format style, the reasonable solution was neither to teach all users the source code on one hand, nor keep producing more and more massive versions that respond to more and more specific needs; the solution was simply to allow the user to intervene, without any predetermination of what that intervention might be. Let me cite one more example, an example which even as I speak may be anachronistic, but the lesson remains. Most data base management programs stipulate a maximum number of characters per record; given the architecture of memory allocation, perhaps some figure is necessary. But there is no reason why a user cannot personally modify that figure if necessary; otherwise, the criterion for inclusion or exclusion of data is the entirely arbitrary one of size. In a bibliography, this would exclude a large number of German works of the eighteenth and nineteenth centuries, and certainly a disproportionately large number of books from German universities.

Computer literacy, then, if it is to provide a sound preparation into the next century, must address principles which, as far as I know, have as yet been neglected in the classroom: what can a computer do and what can it do not; what will it cost to do it; and how flexible can in doing it. Steven Siebert, the author of Nota bene, had not at last report finished his dissertation in Philosophy at Yale. I hope that he has now returned to it, and left Version 3 to someone else. In America and elsewhere, we need as many philosophers as we do programmers.